ECE133A Project

We consider a generalization of the location from range measurements problem in lecture 11. The goal is to determine the location of M points in a plane, given the location of K points with known positions, and inexact measurements of the distances between certain pairs of the points. As an application, the points may represent nodes in a wireless sensor network. Some sensors have been carefully placed or are equipped with GPS receivers, so their location is known. The location of the other sensor nodes has to be determined from the distances to nearby nodes (for example, estimated from the strength of signals received from those nodes).

Nonlinear least squares formulation. The distance measurements can be represented by an undirected graph. We use similar notation as in exercise T12.12.

- The number of vertices in the graph is N. The first $M = N - K$ vertices represent the points with unknown position. We refer to these as the *free nodes*. The last K vertices are the points with known position and are the *anchor nodes*.

- The node positions are denoted by 2-vectors

$$p_1 = (u_1, v_1), \qquad p_2 = (u_2, v_2), \qquad \ldots, \qquad p_N = (u_N, v_N).$$

  The vectors $p_1, \ldots, p_{N-K}$ are the unknowns in the problem. The vectors $p_{N-K+1}, \ldots, p_N$ give the positions of the anchor nodes and are given.

- The edges in the graph indicate the pairs of nodes for which a distance measurement is available. There are L edges, denoted by $(i_1, j_1), \ldots, (i_L, j_L)$. The L distance measurements are

$$\rho_k = \|p_{i_k} - p_{j_k}\| + \varepsilon_k, \quad k = 1, \ldots, L,$$

  where ε_k is measurement error.

Figure 1 shows an example with $N = 30$ nodes and $L = 98$ edges. There are $K = 9$ anchor nodes, shown as squares, and $M = 21$ free points, shown as circles.

To estimate the location of the free nodes we minimize the function

$$\sum_{k=1}^{L} \left(\|p_{i_k} - p_{j_k}\| - \rho_k\right)^2 = \sum_{k=1}^{L} \left(\sqrt{(u_{i_k} - u_{j_k})^2 + (v_{i_k} - v_{j_k})^2} - \rho_k\right)^2. \tag{1}$$

This is a nonlinear least squares problem with variables $u_1, \ldots, u_{N-K}$ and $v_1, \ldots, v_{N-K}$.

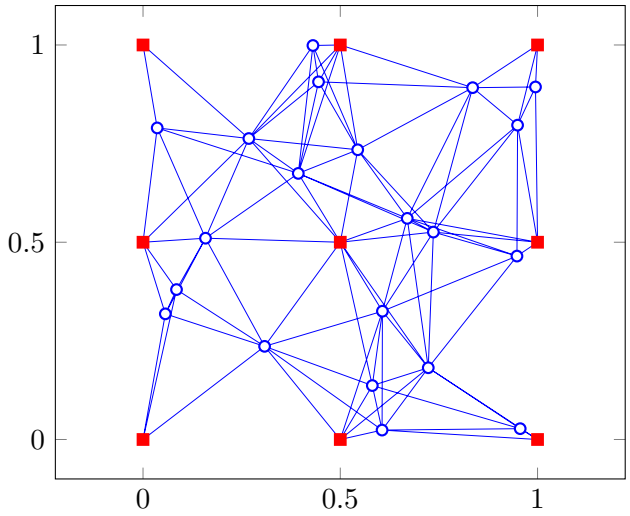

Figure 1: Example with 9 anchor nodes (shown as red squares), 21 free nodes (shown as blue circles), and 98 edges.

Test problems. The MATLAB file `network_loc_data.m` on the class website generates random networks. The calling sequence is

```
[E, pos, K] = network_loc_data(N, R)
```

The first input argument N is the number of nodes in the network. This includes $K = 9$ anchor nodes, positioned as in Figure 1. The $N - 9$ free nodes are placed randomly in the square $[0, 1] \times [0, 1]$. Two nodes are connected by an edge if their distance is less than or equal to R. Figure 1 is an example with $N = 30$ and $R = 0.4$.

The first output argument E is an $L \times 2$ array, specifying the L edges. The two entries of row k are i_k and j_k, the nodes connected by edge k. The output argument `pos` is an $N \times 2$ array with the coordinates of the N nodes. $K = 9$ is the number of anchor nodes.

From the exact positions in `pos` we can compute the exact distances and add random measurement error.

```
L = size(E,1);
d = sqrt(sum((pos(E(:,1),:) - pos(E(:,2),:)).^2, 2));  % exact distances
rho = (1 + .1*randn(L,1)) .* d;   % add about 10 percent error
```

Solving the nonlinear least squares problem with the inexact distance measurements ρ then gives estimates as in Figure 2.

Julia users can use the function defined in `network_loc_data.jl`:

```
E, pos, K = network_loc_data(N, R);
L = size(E,1);
d = sqrt.( sum( (pos[E[:,1],:] - pos[E[:,2],:]).^2, dims = 2) );
rho = (1 .+ .1*randn(L,1)) .* d;
```

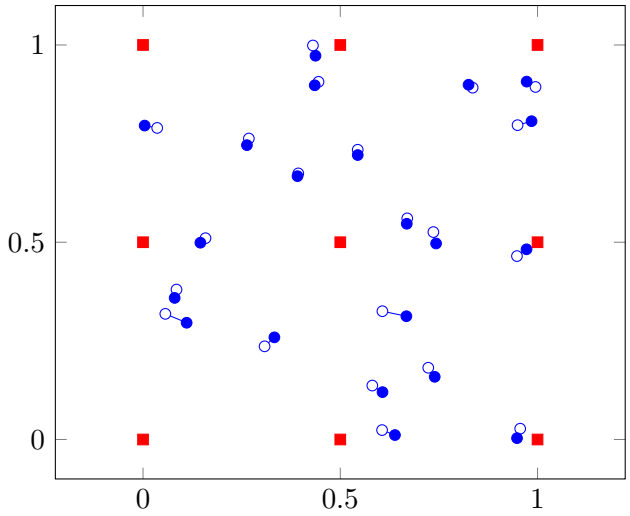

Figure 2: Least squares estimate of the positions. The open circles are the exact positions in Figure 1. The filled circles are the estimated positions.

Assignment. Implement the Levenberg–Marquardt method to minimize (1), following the outline in lecture 11, page 22, or page 392 of the textbook.

- Start the algorithm at randomly generated points (using `rand` in MATLAB or Julia).
- Use the backslash operator to solve the least squares problem in step 2.
- Terminate when $\|\nabla g(x)\| \leq 10^{-5}$.
- Typical values of β_1 and β_2 are $\beta_1 = 0.8$, $\beta_2 = 2.0$.

Report and code. Submit your code and a short report (in pdf).

1. Upload a file `network_loc.m` (MATLAB) or `network_loc.jl` (Julia) with the definition of a function

   ```
   pos_free = network_loc(N, E, pos_anchor, rho)
   ```

   that solves the network node localization problem. Here N, E, and ρ are defined as above, and `pos_anchor` is a $K \times 2$ matrix with the positions of the anchor nodes. The function returns the estimates of the free nodes as an $(N - K) \times 2$ matrix `pos_free`.

   We will test it as follows: in MATLAB,

   ```
   N = 50;  R = 0.4;  s = 0.05;
   [E, pos, K] = network_loc_data(N, R);
   ```

```
pos_anchor = pos(N-K+1:N, :);
L = size(E,1);
d = sqrt(sum( (pos(E(:,1),:) - pos(E(:,2),:)).^2, 2));
rho = (1 + s*randn(L,1)) .* d;
pos_free = network_loc(N, E, pos_anchor, rho);  % your function
```

and in Julia,

```
N = 50;  R = 0.4;  s = 0.05;
E, pos, K = network_loc_data(N, R);
pos_anchor = pos[N-K+1:N, :];
L = size(E,1);
d = sqrt.(sum( (pos[E[:,1],:] - pos[E[:,2],:]).^2, dims = 2));
rho = (1 .+ s*randn(L,1)) .* d;
pos_free = network_loc(N, E, pos_anchor, rho);  # your function
```

2. The report should include a description of the linear least squares problem that you solve at each iteration (*i.e.*, expressions for the entries of the derivative matrix $Df(x)$). Also include results for a typical example (perhaps with 50 or 100 nodes): the estimated and exact positions (as in Figure 2), and plots of the regularization parameter λ and the cost function (1) versus iteration number (as on page 24 of lecture 11).